Why C?

• Now you know basic programming concepts like variables, assignment statements, loops and functions.
• C is a general-purpose versatile programming language.
• C is a relatively "low level" language. It provides:
  – no operations to deal with composite objects, such as character string, arrays, and matrices;
  – no storage allocation facility;
  – no garbage collection.

Hello World

• Get you as quickly as possible to the point where you can write useful programs.
  
  #include <stdio.h>
  main()
  {
    printf("hello, world\n");
  }

• Your program begins executing at the beginning of main().
• main will usually call other functions, some that you wrote, and others from libraries that are provided for you (e.g. stdio).
Basics

• A C program consists of functions and variables.
• A function consists of statements.
• One special function main() – your program begins executing at the beginning of main. This means that every program must have a main somewhere.
• Other functions
  – in libraries.
  – Written by yourself.

Temperature Conversion

• All variables must be declared before they are used. A declaration announces the properties of variables.
• Few variables. Good or not?
• Don’t bury “magic numbers’ in a program.

Character Input and Output

• inout.c
• counting.c
Function

• Functions break large computing tasks into smaller ones, and enable people to build on what others have done instead of starting over from scratch.
• Appropriate functions hide details of operation from parts of the program that don’t need to know about them, and easing the pain of making changes.
• When you need a separate function:
  – Relatively independent tasks.
  – Copy and paste your code.

Function

• run_power.c
Topics

- Chapter 2: types, operators, and expressions
- Chapter 3: Control flow
- Chapter 4: Functions and Program Structure
- Chapter 5: Pointers and Arrays
- Chapter 6: Structures
- Chapter 7: Input and Output

Data Types

- char: a single byte
- int/float/double
- Define constants
  
  ```c
  #define LOWER 100
  ```
  a string constant "hello, world"

  The internal representation of a string has a null character \'\0\' at the end, so the physical storage required is one more than the number of characters written between the quotes.

Length of a string

```c
int strlen(char s[])
{
    int i;
    i=0;
    while (s[i] != \'\0\')
        ++i;
    return i;
}
```
Declarations

• All variables must be declared before use.
  int lower, upper, step;
  char line[1000];

• The qualifier const can be applied to the
declaration of any variable to specify that
its value will not be changed.
  const char msg[] = "warning:";
  int strlen(const char[])

Type Conversion

• atoi()

• An application

• Increment and Decrement Operators
  N = 5;
  X = N++;
  X=++N;

Example

```c
void squeeze(char s[], char c)
{
    int i, j;
    for (i = j = 0; s[i] != '\0'; i++)
        if (s[i] != c)
            s[j++] = s[i];
    s[j] = '\0';
}
```
Example

```c
void strcat(char s[], char t[])
{
    int i,j;
    i = j = 0;
    while (s[i] != '\0')
        i++;
    while((s[i++]=t[j++]=) != '\0')
        ;
}
```

Conditional Expressions

```c
if (a>b)  
    z=a;
else
    z = b;

z = (a>b)? a:b; /* z = max(a,b) */
```

Control flow

- Braces are used to group declarations and statements together into a compound statement.
IF

if (n > 0)
    for (i=1;i<n;i++)
    if (s[i]>0){
        printf("s[%d] is positive",%i);
        return i;
    }
else
    printf("error-n is negative");

Binary Search

• Decide whether a particular value occurs in the sorted array. Assume the elements of the array v are in increasing order.

switch

#include <stdio.h>

int main()
{
    int c, i, nwhite, nother, ndigit[10];
    nwhite = nother = 0;
    for (i = 0; i < 10; i++)
        ndigit[i]=0;
    while ((c = getchar()) != EOF) {
        switch (c) {
        case '0': case '1': case '2':
            ndigit[c-'0']++;
            break;
        case ' ':
        case '
':
        case '	':
            nwhite++;
            break;
        default:
            nother++;
            break;
        }
While

```c
int myatoi(char s[])
{
    int i, n, sign;
    for (i=0;isspace(s[i]);i++)
        ;
    sign=(s[i] == '-') ? -1:1;
    if(s[i] == '+' || s[i] =='-')
        i++;
    for (n = 0; isdigit(s[i]);i++)
        n = 10 * n + (s[i] - '0');
    return sign * n;
}
```

For

```c
void reverse(char s[])
{
    int c, i, j;
    for (i=0, j=strlen(s)-1; i<j; i++, j--)
    {
        c = s[i];
        s[i]=s[j];
        s[j]=c;
    }
}
```

Function

- Revisit “hello world” example.
- Scope Rules
- Header files
- Static variables: static.c
A complicated Example

- Write a program to print each line of its input that contains a particular "pattern" or string of characters.
- While (there is another line)
  if (the line contains the pattern)
    print it
- Three functions (small pieces) are easier to deal with than one big one because irrelevant details can be buried in the functions.
- The chance of unwanted interactions is minimized.

Example

- getline() to get a new line
- strindex() to check whether the pattern is in the line or not.
- We define input and output of the function.
- A minimal function
  dummy()
- How to compile a C program that resides on multiple source files.

Pointers

- A pointer is a variable that contains the address of a variable.
- A typical machine has an array of consecutively numbered or addressed memory cells that may be manipulated individually or in contiguous groups.
- A char variable takes one cell.
- A point is a group of cells (2 or 4) that can hold an address.
& and *

- P=&C assigns the address of c to the variable p.
- & can be applied to any type of variable that is in the memory.
- * is the dereferencing operator – when applied to the point, it accesses the object the pointer points to.

Using pointers

- See an example pointer1.c
- A pointer is constrained to point to a particular kind of object: every pointer points to a specific data type.
- If ip points to the integer x, then *ip can occur in any context where x could:
  
  ```
  *ip = *ip + 10;
  ```

Function Arguments

- If C passes arguments to functions by value, there is no direct way for the called function to alter a variable in the calling function.
  
  ```
  void swap(int x, int y){
    int temp;
    temp = x;
    x = y;
    y = temp;
  }
  ```
- swap cannot affect the argument a and b in the routine that called it. The function above only swaps copies of a and b.
swap(&a,&b)

```c
void (int *px, int *py){
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

Array

```c
int a[10];
int *pa;
pa = &a[0]; /* pa and a have identical values */
x = *pa;
pa + i is the address of a[i], and *(pa+i) is the contents of a[i].
The above is true regardless of the type or size the variables in the array a.
Adding 1 to a pointer causes it to point to the next object.
In evaluating a[i], C converts it to *(a+i) immediately; the two forms are equivalent.
Similarly, if pa is a pointer, pa[i] is identical to *(pa+i).
```

Array

- When an array name is passed to a function, what is passed is the location of the initial element. Within the called function, this argument is a local variable, and so an array name parameter is a pointer – a variable containing an address.

```c
int strlen(char *s)
{
    int n;
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}
```

- As formal parameters in a function definition, char s[] and char *s are equivalent.
Character pointers and functions

- char amessage[] = "now is the time"
- char *pmessage = "now is the time"
- amessage is an array, just big enough to hold the sequence of characters and \"0\" that initialized it. Individual characters within the array may be changed but amessage itself will always refer to the same storage.
- pmessage is a pointer pointing a string constant; the pointer may subsequently be modified to point to elsewhere.

Example

```c
void strcpy(char *s, char *t)
{
    while ((*s++ = *t++) != \"0\")
}
```

Pointer version?

Pointer Array

- char *lineptr[100]
- What is the difference between int a[10][20] and int *b[10]?
- What is the difference between int (*daytab)[13]; int *daytab[13];
Multidimensional Arrays

- $a[3][4]$ and $b[3][4]$ can both syntactically legal references to a single int.
- But $a$ is a true two-dimensional array; 200 int-sized locations have been set aside, and the conventional rectangular subscript calculation $20 \times \text{row} + \text{col}$ is used to find the element $a[\text{row}][\text{col}]$.
- For $b$, however, the definition only allocates 10 pointers and does not initialize them. If each element of $b$ does point to a twenty-element array, then there will be 200 ints set aside, plus ten cells for the pointers.
- The important advantage of the pointer array is that the rows of the array may be of different lengths. That is, each element of $b$ need not to a 20-element vector; some may point to two, some to fifty.
- e.g. `char *name[] = {"January", "February", "March"};`

Command-line Arguments

```c
main(int argc, char *argv[])
```

A new version of grep.c

Input and Output

- `int getchar()`
- From the keyboard or from a file
- Output the result into a file
- `printf()`
Scanf()

- Reads characters from the standard input, interprets them according to the specification in format, and stores the results through the remaining arguments.
- E.g. input line: 25 Dec 2006
  int day, year;
  char month[20];
  scanf("%d %s %d", &day, &month, &year);
* Scanf ignores blanks and tabs in its format string.

File Access

- File pointer
  FILE *fp;
- FILE *fopen(char *name, char *mode);
- int getc(FILE *fp);
  int putc(int c, FILE *fp);
- int fscanf(FILE *fp, char *format,…)
  int fprintf(FILE *fp, char *format, …)